CS451 - Assignment 7
Naive Bayes or Intelligent Bayes?

Due: Sunday, November 3rd by 11:59pm

http://xkcd.com/656/

For this assignment we're going to be adding another multi-class classifier to our collection, the Naive Bayes classifier. In addition, we're going to play with generating ROC curves.

1 Starter

For this assignment there's no starter! You can either grab the starter from assignment 6 or just copy over your solution to assignment 6 and start from there.

2 Specifications

Implement a class called `NBClassifier` that implements the `Classifier` interface. Your NB classifier should treat each feature as a binary feature, that is your probability distributions $p(x_i|y)$ are only over two values, *positive* and *negative*. We will treat any feature that is both available AND non-zero as *positive* and any feature that is either not available or zero as *negative*.

Your class should:

- have a zero parameter constructor

- include a `setLambda` function that allows you to set the λ regularization/smoothing parameter

- support two different ways of calculating the probability of examples. The mathematically correct way of implementing Naive Bayes is to calculate $p(x_1, x_2, ..., x_m, y)$ as:

$$p(x_1, x_2, ..., x_m, y) = p(y) \prod_{i=1}^{m} p(x_i|y)$$

  Specifically, if a features is present, then you're multiplying by $p(x_i = \text{positive } |y)$ and if it is not present then you're multiplying by $p(x_i = \text{negative } |y) = 1 - p(x_i = \text{positive } |y)$.

  In some domains, however, if the examples are very sparse (e.g. text) this can be very expensive and, in some cases, non-intuitive. Another variant is often considered in these domains is calculating $p(x_1, x_2, ..., x_m, y)$ using only the features that are present/positive:

$$p(x_1, x_2, ..., x_m, y) = p(y) \prod_{i=1}^{m} 1[x_i = \text{positive}] p(x_i|y)$$

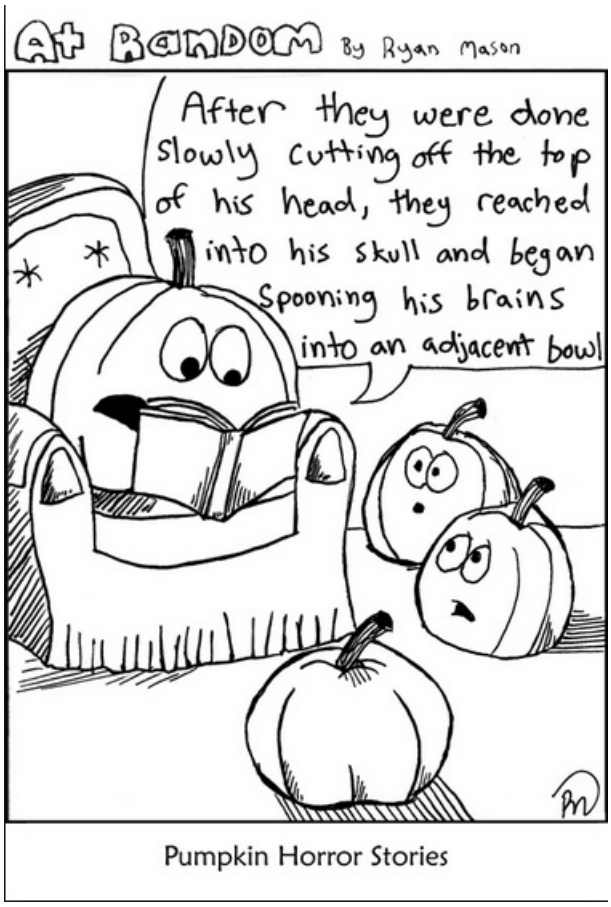
  Your implementation should support both ways of classifying an example. A few things to note: 1) the training method and method for calculating individual feature probabilities is exactly the same between these two approaches and 2) when only considering positive features, for efficiency reasons, you should only iterate over the features available for the example and NOT over all features. Notice that the `DataSet` class has a function that tells you all the features that are available and the `Example` class has a function that tells you all features available for a given example.

- include a `setUseOnlyPositiveFeatures` function that takes a `boolean` and chooses between the different classification variants just described. If it is set to `true` then is should only use the positive features. By default, this should be set to `false`, that is to use all features.

- include the following functions:

```
public double getLogProb(Example ex, double label)
public double getFeatureProb(int featureIndex, double label)
```

  The first should return the log probability of the example with the label under the current trained model and the second should give $p(x_i|y)$. These will help me test your model's correctness.



http://www.atrandomcomics.com/

3 Hints/Advice

- When dealing with multiplying probabilities, you should almost always use log probabilities and not probabilities since underflow often becomes a problem. Specifically, instead of classifying as:

$$label = argmax_{y \in labels}\ p(x_1, x_2, ..., x_m, y)$$

you should use:

$$label = argmax_{y \in labels}\ \log(p(x_1, x_2, ..., x_m, y))$$

which for the all features case then becomes:

$$label = argmax_{y \in labels}\ log(p(y)) + \sum_{i=1}^{m} log(p(x_i|y))$$

- You should NOT regularize/smooth the label probabilities $p(y)$
- When regularizing/smoothing the feature probabilities your probabilities will be:

$$\hat{p}(x_i|y) = \frac{\text{count}(x_i, y) + \lambda}{\text{count}(y) + \text{num\_features} * \lambda}$$

  where `num_features` is the total number of features available in the training set.
- In the `ml.utils` package there are a couple of classes that might be useful for this assignment. In particular, take a look at the `HashMapCounter` class.
- Your `confidence` function should return the log probability of the most likely label (i.e. the label chosen for that example).
- Make sure to set it up so that each time you call the `train` method all of your data gets wiped clean and you start from scratch. This will allow you to train the same classifier multiple times on different data.
- During training you can either just store the raw counts or you can collect the counts and then calculate and store the probabilities. There are advantages to either approach. Most often people choose the first (storing raw counts) for two reasons: 1) you can change the λ value without having to retrain and 2) the model can be stored as integers instead of doubles, creating some memory savings. That said, storing the probabilities does save us some computation during classification. Either implementation is fine for this assignment.

4 Experiments

In a file called `experiments` (pick some reasonable file type) include the analysis below (explicitly number each part in your writeup). For all the experiments use the *wine data set*.

1. What is the optimal lambda for the NB classifier using all features? Show your experimental data for how you found this.
2. What is the optimal lambda for the NB classifier using only positive features? Show your experimental data for how you found this.
3. Which version of the NB classifier is better (all features vs. positive features). Include one or two sentences that describe how you came to this conclusion.
4. One of the uses of classifier confidence is to allow you to improve the accuracy of the classifier by only labeling those examples above a given confidence threshold. If your confidence measure is a good predictor, you can get an increase in accuracy with only a small number of examples not getting labeled.

   To explore this, create a graph the plots the accuracies on the y-axis versus the confidence thresholds on the x-axis based on the first split of the data using a 10-fold cv split. To do this you'll need to:

(a) train the NB classifier

(b) for each example, get the label prediction and the confidence for that prediction

(c) sort these pairs in decreasing order by confidence

(d) working from the most confident to the least, calculate the accuracy if you placed the confidence threshold at the current example. If your data is sorted by confidence, this should be easy to do in a single pass over the data by keeping track of how many you've seen already that are correct. This will give you a (accuracy,confidence cutoff) point for each test example which you can then plot.

5 When You're Done

Make sure that your code compiles, that your files are named as specified and that you have followed the specifications exactly (i.e. method names, number of parameters, etc.).

Create a directory with your last name, followed by the assignment number, for example, for this assignment mine would be `kauchak7`. If you worked with a partner, put both last names.

Inside this directory, create a `code` directory and copy all of your code into this directory, maintaining the package structure.

Finally, also include your `writeup` file.

`tar` then `gzip` this folder and submit that file on the submission page on the course web page.

Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file
- Each class and method should have an appropriate JavaDoc
- If anything is complicated, it should include some comments.

There are many possible ways to approach this problem, which makes code style and comments very important here so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.